LR Parsers

(1) One type of bottom-up parser.

(2) It is based on a concept called LR(k) parsing:

"L" - left to right scanning of input.

"R" - Constructing a rightmost derivation in reverse

'k" - No. of ~~lookahead~~ input symbols of lookahead that are used in making parse decisions.

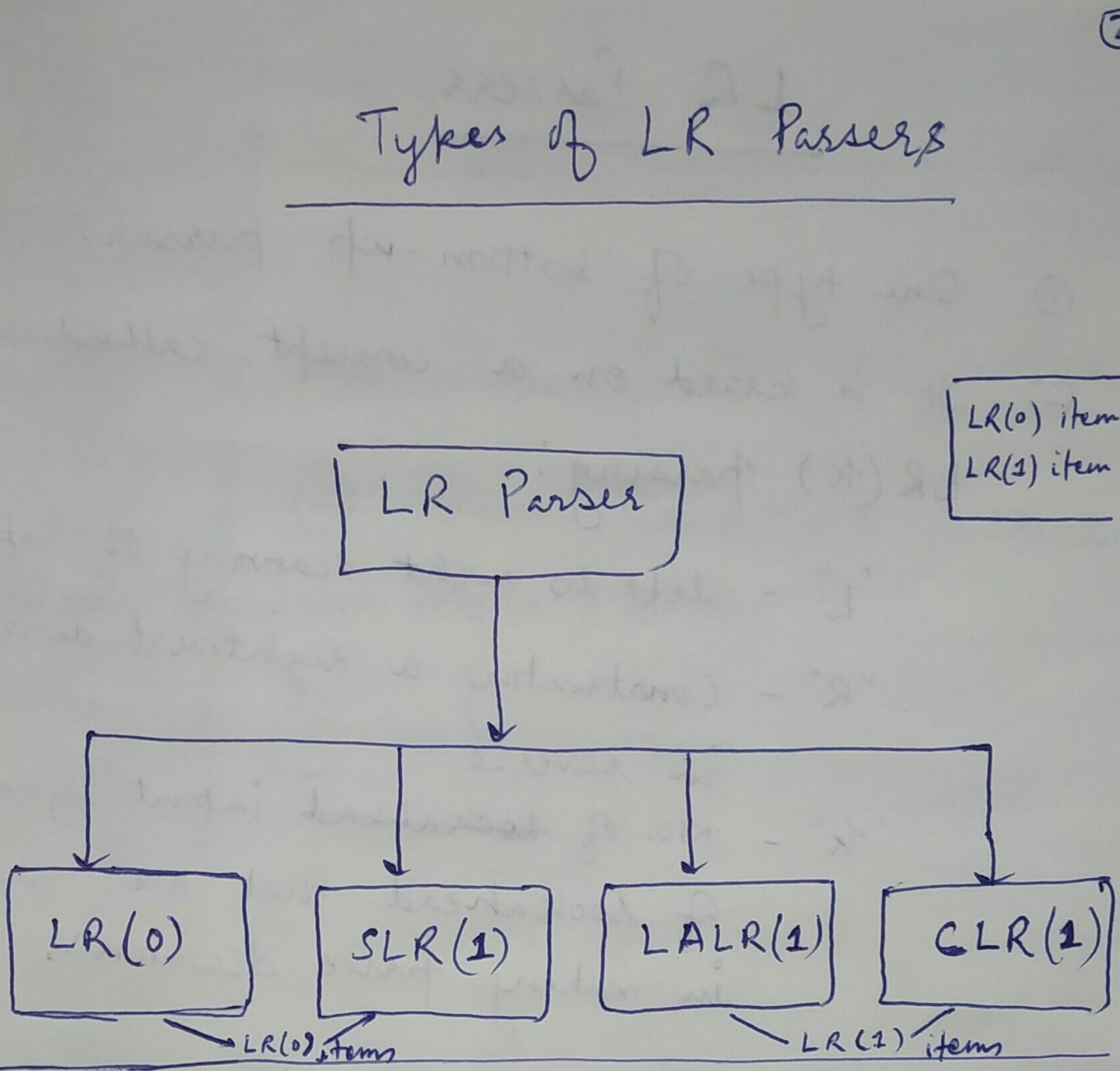
(3) We shall only consider LR parser with $k \leq 1$ here.

→ When (k) is omitted, k is assumed to be one.

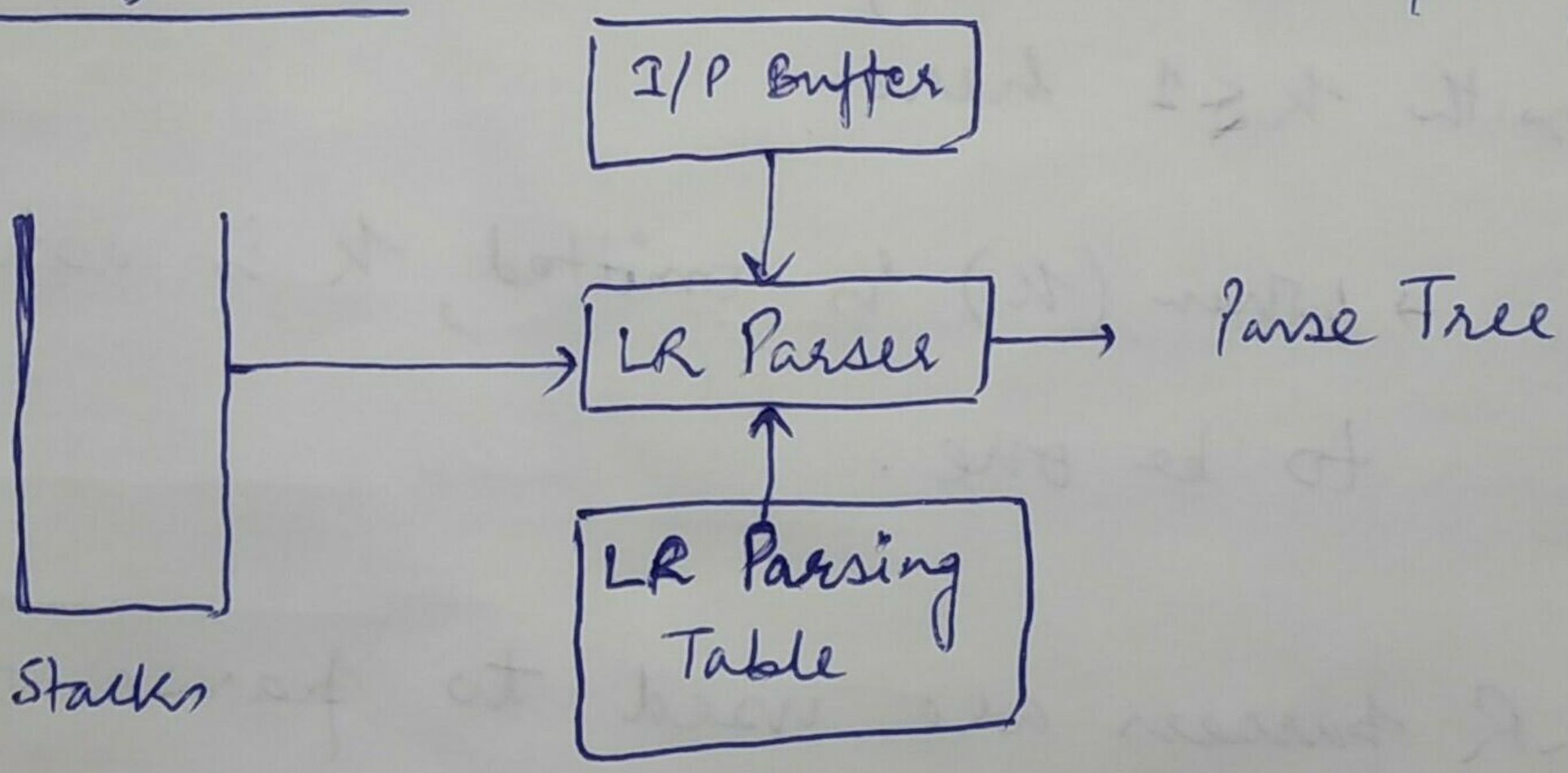
(4) LR parsers are used to parse the large class of Context Free Grammers (CFGs)

Types of LR Passers

LR(0) item
LR(1) item



⇒ LR Algorithm requires input, output, stack, and parsing table



<Block Diagram of LR Parser>

Why LR Parsers

(1) LR parsers can handle a large class of context-free grammars.

(2) The LR parsing method is a most general non-back tracking shift-reduce parsing method.

(3) An LR parser can detect the syntax errors as soon as they can occur.

(4) LR grammars can describe more language than LL grammars.

Drawbacks of LR parsers

(1) It is too much work to construct LR passer by hand. It needs an automated passer generator.

(2) If the grammar contains ambiguities ~~ambiguities~~ or other constructs then it is difficult to parse in a left-to-right scan of the input.

Steps to solve LR(0) items

(1) Augment the given grammar

(2) Draw Canonical Collection of LR(0) items / DFD

(3) Number the Production

(4) Create the Parsing table

(5) Stack implementation

(6) Draw Parse Tree

~~Grammar~~

Using LR(0) contruct the parse tree for "ccdd", where grammar is

$$E \rightarrow BB$$
$$B \rightarrow cB \mid d$$

Step 1) Augment the given Grammar

(If G is a grammar with start symbol S, then G' is the augmented Grammar for ~~S' → S~~ G, with a new start symbol S' and production S' → S.)

$$E' \rightarrow E$$
$$E \rightarrow BB$$
$$B \rightarrow cB / d$$

Step 2) Draw Canonical collection of LR(0) item/ DFD/ DFA

LR(0) item

Any production with . in the beginning is called LR(0) item. for eg: in G'

E' → .E for E' → E, and then add the productions followed by goto marked closure.

I0

E' → .E

E → .BB

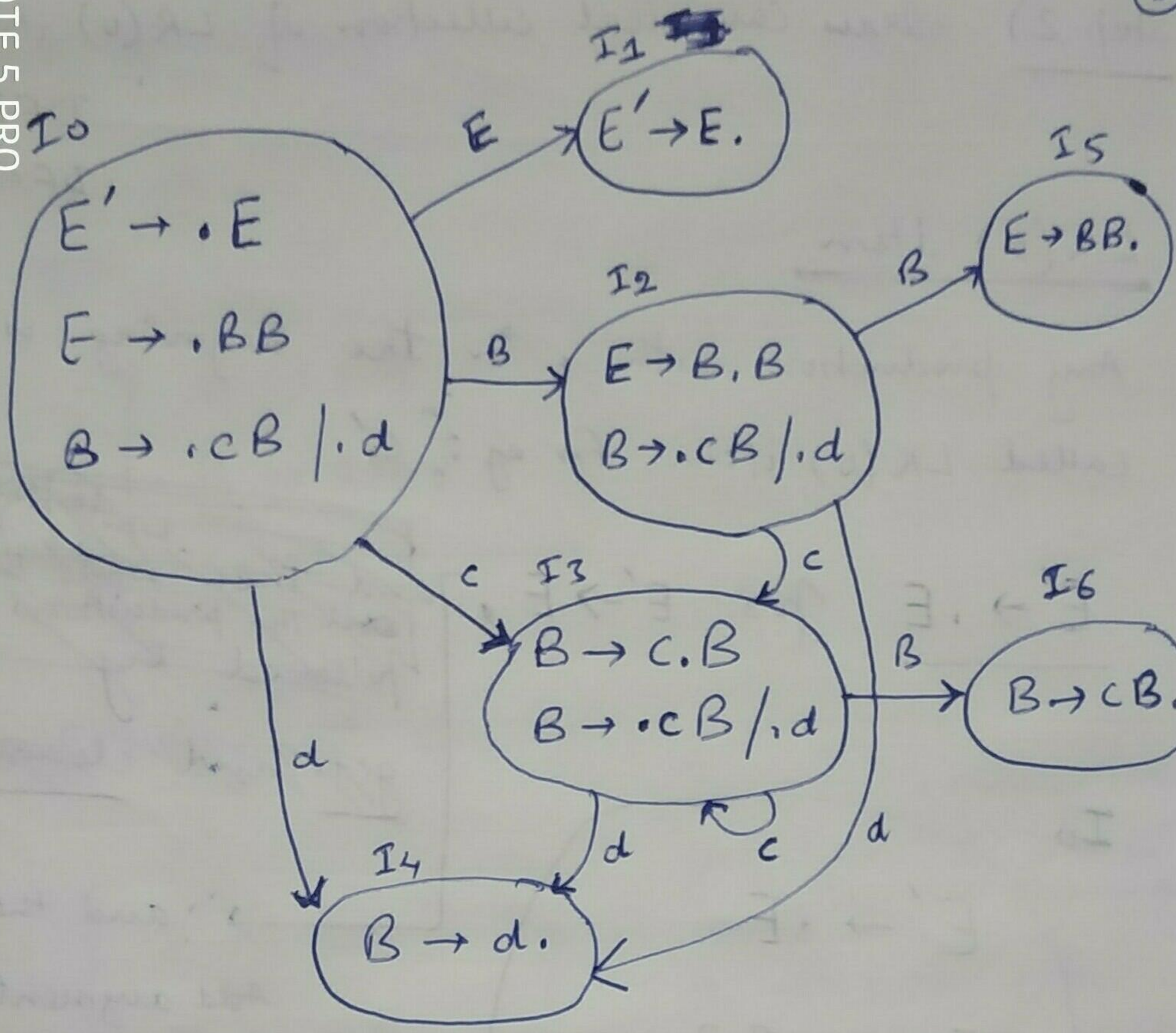
B → .cB | .d

→ and then Add augment Production and insert '.' symbol at the first position for every production in G.

Then followed by goto and closure

↳ (action to be performed)

(8)



Q1 What is closure?

Q2 What is goto?

Q3 What is final item?

↳ whenever . is in the right hand side, and we cannot move further. for eg.

E → BB. It means now we can reduce it.

E
^
BB

< Canonical collection of LR(0) item >
and their states

Step 3) Number the production

(Total productions in original grammar)

$E' \rightarrow E$

$E \rightarrow \underline{BB}$ ①

$B \rightarrow \underline{cB}$ ② / $\underline{d}$ ③

Step 4) Create the Parsing Table

State	Action			Goto	
	c	d	$	E	B
I_0	S_3	S_4		1	2
→ I_1			Accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
→ I_4	R_3	R_3	R_3		
→ I_5	R_1	R_1	R_1		
I_6	R_2	R_2	R_2		

Step 5) Stack Implementation

Grammar

$E' \rightarrow E$

$E \rightarrow BB$ (1)

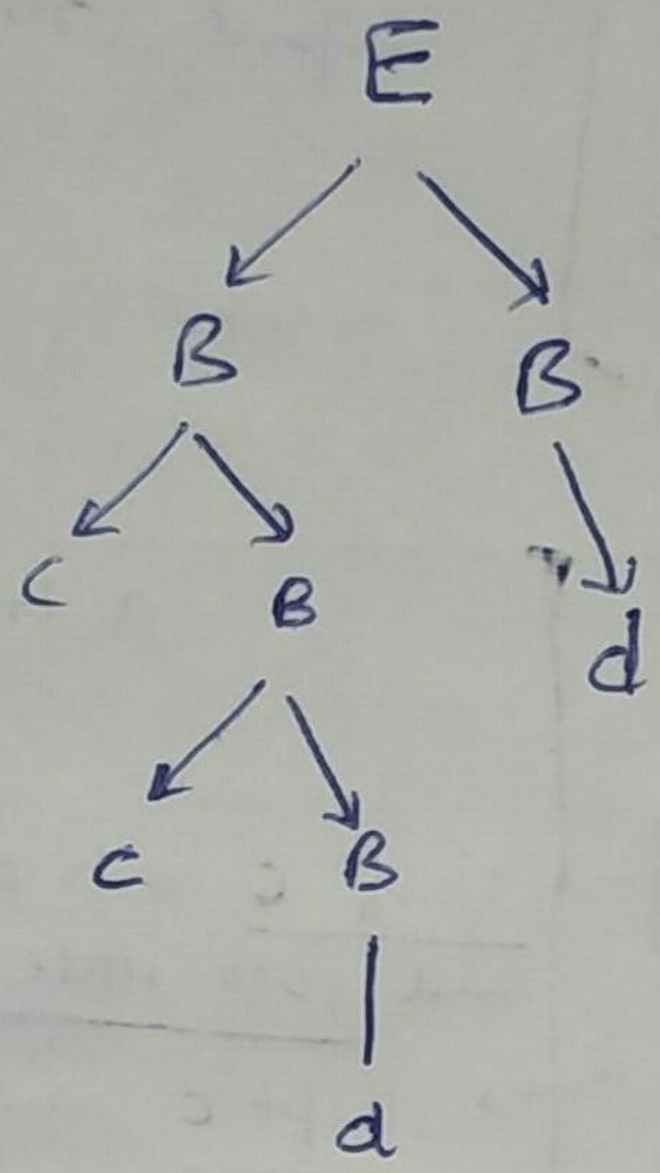
$B \rightarrow cB$ (2) / d (3)

input string: c c d d

Initially

Stack	Input	Action	Remarks
$0	c c d d $	→ shift c onto the stack and goto state 3	state 0 input c
$0c3	c d d $	→ shift c and goto S3	state 3 input c
$0c3c3	d d $	→ shift d and goto S4	state 3 input d
$0c3c3d4	d $	→ Reduce (R3), B → d	Reduce ↓ (1) Reduce (d4) with B (2) Then B what is there in B (I3) = 6 So it will become B 6
$0c3c3B6	d $	→ Reduce (R2), B → cB	
$0c3B6	d $	→ Reduce (R2), B → cB	
$0B2	d $	→ shift d and goto S4	
$0B2d4	$	→ reduce (R3), B → d	
$0B2B5	$	→ reduce (R1), E → BB	
$0E1	$	→ Accept	
✓	✓		

Step 6) Draw the Parse Tree



SLR(1) [Simple LR]

→ Every thing same except Passing Table

State	Action			Goto	
	c	d	$	E	B
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	r_3	r_3	r_1		
I_5			r_1		
I_6	r_2	r_2	r_2		

I_4 $\frac{B \to d.}{(B \to d)}$, Follow(B) = {c, d, $}

I_5 $\frac{E \to BB.}{(E \to BB)}$, Follow(E) = {$}

I_6 $\frac{B \to cB.}{(B \to cB)}$, Follow(B) = {c, d, $}

Step 3

$E' \to E$ ①

$E \to BB$

$B \to cB / d$

② ③

First and Follow in Parsing

NT → T

A → ab

A → aB | ∈

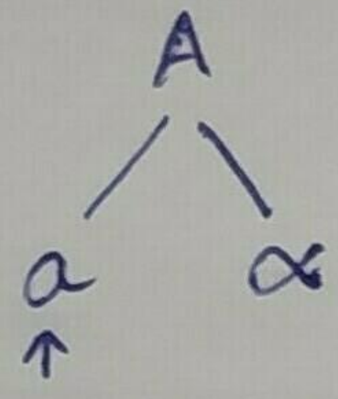
or

A → aB

A → ε

→ we compute First and Follow of NT's

① $A \to a\alpha$



First of A = {a} // First leftmost terminal or terminal

or

First (A) = {a}

② $A \to \varepsilon$

First (A) = {ε}

③ $A \to B$

$B \to C$

$C \to id$

A → B → C → id

First (A) = {id}

Q. on First

$A \rightarrow a$

$B \rightarrow (c) \mid id$

$c \rightarrow *T \mid \epsilon$

First(A) = {a}

First (B) = {(, id} (→ bracket

First (c) = {* , ε}

Follow →

① $A \rightarrow \alpha B \beta$

$\beta \rightarrow a$

Follow (B) = {a} // B is followed by β if non-terminal then first of it is follow

Ex

$A \rightarrow TE'$

$E' \rightarrow +f \mid id$

Follow (T) = {+, id}

② $A \rightarrow \alpha B \beta$

$\beta \rightarrow \varepsilon$

// there will be no epsilon in follow

Follow (B) → {All follow of A}

Ex

$E \rightarrow TE'$

follow (E) = {$,)} // assuming

follow (E') = {$,)} // follow of E

(3) $A \to \alpha B \beta$

$\beta \to a \mid \varepsilon$

then what will be follow(B)

Follow (B) = { a, all follow of A }

Q.1

$E \to TE'$

$E' \to +TE' / \varepsilon$

$T \to FT'$

$T' \to *FT' / \varepsilon$

$F \to (E) / id$

Q.2

$S \to iEtSS' / a$

$S' \to eS \mid \varepsilon$

$E \to b$

Symbol (NT)	First	Follow	
E	(, id	), $	
E'	+, ε	), $	// follow of E
T	(, id	+,), $	// First(E') = + ε ⇒ E' E' is last follow of E
T'	*, ε	+,), $	
F	(, id	*, +,), $	

Q Check whether a Grammar is LR(0) or not?

$G \Rightarrow E \to T \overset{①}{+} E \;/\; \overset{②}{T}$

$T \to id \overset{③}{}$

State	Action id	+	$	Goto E	T
0	S_3			1	2
1			accept		
2	R_2	S_4/R_2	R_2		
3	R_3	R_3	R_3		
4	S_3			5	2
5	R_1	R_2	R_3		

After creating the DFA/DFD or Parsing table

check for conflicts (~~shift/shift~~ ~~or~~ shift-reduce or reduce-reduce)

if any then G is not a ~~LR(0) item~~ or LR(0)